Migración de Procesos: Alternativas a sus Desventajas

Javier Echaiz Rafael B. García Jorge R. Ardenghi

Laboratorio de Investigación de Sistemas Distribuidos (LISiDi)
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
e-mail: {je, rbg, jra}@cs.uns.edu.ar

Resumen

La migración de procesos consistente en tranferir un proceso entre dos máquinas. Las posibilidades que ofrece incluyen: distribución dinámica de carga, tolerancia a fallas, mejor administración del sistema y localidad de acceso a los datos. Sin embargo, a pesar de lo interesante de estas posibilidades y del esfuerzo investigativo invertido en este tema, no se logró aún que la migración de procesos sea utilizada popularmente. Gracias al creciente desarrollo de los sistemas distribuidos en general y de los sistemas operativos distribuidos en particular, la migración de procesos está recibiendo atención no solo en el ámbito académico sino también en el comercial. La tendencia de utilizar redes de workstations en lugar de supercomputadoras y el rol que está ocupando la World Wide Web, hacen suponer que la migración logrará una importante difusión en un futuro próximo.

En este trabajo presentamos el estado del arte y las cuestiones concernientes a la migración de procesos. Además trataremos de identificar los impedimentos que atentan contra la popularidad de la migración y que técnicas se pueden emplear para solucionarlos.

Palabras Clave: migración de procesos, sistemas distribuidos, sistemas operativos distribuidos, balance de carga.

1 Introducción

Un proceso es una abstracción del sistema operativo que representa una instancia de ejecución de un programa. *Migrar un proceso* es transferirlo de una computadora hacia otra durante su ejecución. Existen muchos sistemas operativos que implementan migración de procesos, entre ellos se encuentran MOSIX, V, Accent, Mach y OSF/1. Además algunos sistemas proveen mecanismos que permiten detener la ejecución de los procesos de una máquina y continuarlos en el mismo estado en otra, como por ejemplo en Condor y LSF. La migración de procesos permite:

- **distribución dinámica de carga**, migrando procesos desde nodos sobrecargados hacia nodos menos cargados,
- **tolerancia a fallas**, migrando procesos pertenecientes a nodos que puedan haber sufrido una falla parcial,

- **mejor administración del sistema**, migrando procesos desde nodos que están por apagarse y que de otra manera no estarían disponibles,

- **localidad de acceso a los datos**, migrando procesos hacia nodos que se encuentren más cerca de los datos.

A pesar de lo interesante que lucen estas metas, la migración de procesos todavía no ha alcanzado un uso generalizado. Una de las razones que fundamentan este hecho es la complejidad de agregar migración transparente a sistemas que fueron originalmente diseñados para funcionar *stand-alone*. Otra razón tiene que ver con el aspecto comercial de los sistemas operativos: no hay un buen argumento para soportar la migración. La propuesta detener-continuar ofrece una solución de compromiso pues puede implementarse en sistemas débilmente acoplados, si se restringe el tipo de procesos que pueden migrar.

Sin embargo, la migración de procesos sigue siendo un área activa para la investigación. Probablemente esto se deba principalmente al potencial que ofrece a los diferentes tipos de aplicaciones y a la atracción académica que presentan los problemas no resueltos completamente.

En los últimos tiempos se incrementó el desarrollo de los sistemas distribuidos en general y los sistemas operativos distribuidos en particular, esto hizo que la migración de procesos fuese de interés no solo para los investigadores sino también para los desarrolladores. Los sistemas de alta performance que antaño conformaban las supercomputadoras estan siendo reemplazados por redes de workstations (NOWs) [1] y sistemas distribuidos de gran escala, es de esperarse entonces que la migración logre un rol más importante y que eventualmente obtenga mayor aceptación.

2 Nociones Básicas

Esta sección presenta brevemente nociones básicas sobre migración de procesos, especifícamente bjetivos, algoritmos de migración, manejo de información de carga, scheduling distribuido y alternativas a la migración.

2.1 Objetivos

Los objetivos de la migración de procesos están fuertemente relacionados con el tipo de aplicación que usa la migración, como se describe en la próxima sección. Los objetivos de la migración de procesos son:

- **Lograr mayor poder de procesamiento** es importante si se emplea la migración para distribuir la carga. La migración es particularmente importante en los algoritmos de scheduling distribuido *inicidos por el receptor*, donde un nodo poco cargado anuncia que está disponible e inicia la migración de procesos de un nodo sobrecargado. Esta fue la meta de varios sistemas distribuidos, tales como Locus, MOSIX y Mach. La distribución de carga también depende del manejo de la información de carga y del scheduling distribuido, como veremos en las secciones 2.3 y 2.4. Una variante de esta meta es aprovechar el poder de cómputo de las workstations libres migrandoles procesos hasta que el dueño de dicha workstation regrese, e.g. Sprite.

- **Explotar la localidad de los recursos** es deseable para la migración cuando es más eficiente acceder a recursos locales en vez de remotos. Si se mueve un proceso hasta la otra punta de un canal de comunicación la comunicación remota se vuelve local y en consecuencia se mejora significativamente la performance. También es posible que el recurso no esté disponible remotamente, por ejemplo cuando existen diferentes semánticas para los accesos locales y remotos.

- **Compartir recursos** mediante el uso de la migración brinda a un nodo específico hardware adicional, e.g. memoria o algún otro recurso valioso (workstations no utilizadas).

- La **tolerancia a fallas** mejora debido a la migración si se da el caso en el que un nodo sufre una falla parcial, o cuando es probable que ocurran distinos tipos de fallas (red, dispositivos) durante la ejecución prolongada de aplicaciones. En este contexto se puede utilizar migración junto con la técnica de checkpoints, e.g. en Condor y en Utopia. En sistemas de gran escala, donde es probable que algunos de los sistemas falle la migración puede ser beneficiosa, e.g. en Hive y en OSF/1.

- La **administración de sistemas** se simplifica si los cómputos que llevan largo tiempo de cómputo son transferidos a otros nodos. Por ejemplo se puede migrar un proceso de un nodo que será bajado y luego traerlo nuevamente cuando esté nuevamente disponible.

- La **computación móvil** también aumenta la demanda de la migración pues por ejemplo un usuario puede desear migrar un proceso desde un nodo de la empresa a su computadora móvil antes de retirarse y al dia siguiente migrarlo nuevamente hacia el primer nodo.

2.2 Algoritmo de migración

Si bien existen diferentes diseños e implementaciones de algoritmos de migración, la mayoría de ellos puede sintetizarse en los siguientes pasos:

1. **Se envía un pedido de migración a un nodo.** Luego de la negociación la migración es aceptada.

2. **Un proceso se desconecta de su nodo origen**. Esto es, suspendender su ejecución, declararlo en estado de migración y redirigir temporalmente la comunicación, como se describe en el paso siguiente.

3. **La comunicación se redirige temporalmente** encolando los mensajes destinados al proceso migrado y entregándoselos luego de la migración. Este paso continúa en paralelo con los pasos 4, 5 y 6 mientras sigan llegando mensajes. Una vez que los canales de comunicación están habilitados después de la migración (como resultado del paso 7) el proceso migrado se hace público al mundo exterior.

4. **Se extrae el estado del proceso**, incluyendo lo almacenado en memoria, estado del procesador (registros), estado de comunicación (e.g. archivos abiertos y canales de mensajes) y contexto del kernel, siendo estos dos últimos dependientes del S.O. Parte del estado interno del S.O. no es transferible. El estado del procesador es típicamente retenido en el nodo fuenta hasta el final de la migración y en algunos sistemas se lo mantiene allí aún después de completarse la misma. Las dependencias relativas al procesador, tales como los contenidos de los registros y del stack deben eliminarse si se trata de migración heterogénea.

5. **Se crea una instancia del proceso destino** donde se importará el estado transferido. Esta instancia se activará recién cuando se cuente con suficiente estado transferido desde la instancia del proceso fuente. Después de la transferencia la instancia destino se promueve a proceso normal.

6. **El estado es transferido e importado en una nueva instancia** en el nodo remoto. No se necesita transferir todo el estado, parte del mismo puede traerse luego de la migración.

7. **Se deben mantener redireccionamientos** hacia el proceso migrado para poder comunicarse con el mismo. Esto puede llevarse a cabo registrando la ubicación actual en el nodo *home* (e.g. en Sprite), buscando el proceso migrado (e.g. en el Kernel V, a nivel del protocolo de comunicación) o redireccionando mensajes desde los nodos visitados (e.g. en Charlote). Este paso también habilita los canales de comunicación del proceso destino y concluye con el paso 3 al redirigir la comunicación permanentemente.

8. **La ejecución de la nueva instancia prosigue** cuando se importó suficiente información de estado. Una vez transferido todo el estado, la instancia fuente puede eliminarse. Con este paso concluye la migración del proceso.

2.3 Manejo de la Información de Carga

Para determinar que proceso migrar (y si se justifica su migración) se deben caracterizar los procesos locales y los recursos tanto locales como los de los nodos remotos. A esta tarea, conocida como manejo de la información de carga, le conciernen las siguientes tres preguntas:

¿Cuál es la información de carga y cómo se la representa? La carga del nodo generalmente se representa con uno o más de los siguientes índices de carga: utilización del CPU, longitud de la cola de procesos en espera de ejecución, número de procesos en ejecución, número de procesos en background, comunicación, utilización del disco, consumo de memoria (física y virtual) y frecuencia de interrupciones.

¿Cuándo se recolecta y distribuye la información de carga? Estas operaciones pueden ser periódicas (cada 1 segundo o más) o basadas en eventos (creación, terminación o migración de un proceso). La frecuencia de la distribución de la información es generalmente menor que la de la recoleción, i.e., es promediada para evitar la inestabilidad [2]. La frecuencia también depende del costo involucrado en estas operaciones: a menor costo menor período; a mayor costo menos frecuentemente se puede distribuir la información de carga.

¿Cuánta información debe transferirse? Puede ser el estado completo, pero generalmente se mueve un subconjunto del mismo con el fin de minimizar los costos de transferencia y maximizar la escalabilidad.

Se pueden realizar dos observaciones a partir de esta última pregunta. La primera es que una cantidad pequeña de información puede lograr un aumento significativo en la performance, y esto es tan válido para distribución de carga en general como para migración de procesos. La segunda observación es que el tiempo de vida de un proceso puede emplearse en mecanismos de distribución de carga. El punto está en saber que tan viejo debe ser el proceso como para que valga la pena migrarlo. En [3] exploraron este tema y obtuvieron una distribución del tiempo de vida de un proceso que les permitió correlacionar el tiempo de vida con la edad del proceso y derivar un criterio de costo para la migración. Otros estudios involucran el uso de estadísticas [4] o de técnicas de inteligencia artificial [5].

2.4 Scheduling Distribuido

El scheduling distribuido utiliza la información provista por el módulo de manejo de información de carga. El objetivo principal es determinar *que* proceso migrar, *cuando* y *hacia donde*. La política de activación responde la pregunta de *cuando* migrar. El scheduling se activa periódicamente o por eventos. Luego de la activación se inspecciona la carga y si está por encima o por debajo de un umbral se actúa según la estrategia seleccionada. La política de selección provee la respuesta a la pregunta de *que* proceso migrar. Se inspeccionan los procesos y algunos de ellos se eligen (según algún criterio) para ser migrados. *Donde* migrar depende del algoritmo de ubicación, el cual selecciona el nodo remoto según la información disponible.

2.5 Alternativas a la Migración de Procesos

La complejidad de la implementación y el costo de invocar el mecanismo de migración de procesos promueven el desarrollo de implementaciones alternativas a la migración [6].

La **ejecución remota** es la alternativa más frecuentemente empleada. Puede ser tan simple como invocar código en un nodo remoto o puede involucrar transferir código hacia el nodo remoto y heredar parte del entorno del proceso, e.g. variables y archivos abiertos. La ejecución remota es generalmente más rápida que la migración de procesos debido a que no necesita transferir un estado de proceso potencialmente grande.

Sin embargo, la ejecución remota también presenta desventajas. La creación de la instancia remota es permitida solamente durante la creación del proceso, a diferencia de la migración, la cual permite que el proceso migre en cualquier momento. Permitir que un proceso se ejecute en el nodo fuente por algún tiempo es útil en ciertos casos, pues los procesos con cortos tiempos de vida (y que por ende no vale la pena migrar) son automáticamente filtrados. Además cuanto mayor sea el tiempo de ejecución de un proceso mayor será la cantidad de información disponible acerca de su comportamiento, como por ejemplo con qué procesos se comunica. Ejemplos de sistemas que emplean ejecución remota son Sprite, Amoeba, Hive y Utopia.

La **clonación de procesos** es útil cuando el proceso hijo hereda estado del proceso padre. En general la clonación se lleva a cabo mediante un mecanismo de *fork* remoto. El fork remoto, seguido de la terminación del proceso padre es similar a la migración. La clonación presenta la misma complejidad que la migración pues se hereda la misma cantidad de estado (e.g. archivos abiertos y espacio de direccionado). En el caso de la migración el padre es terminado pero en el de la clonación tanto el padre como el hijo pueden seguir accediendo al mismo estado, introduciendo un estado compartido distribuido, el cual es complejo y costoso de mantener. Existen varios sistemas que emplean el fork remoto, uno de ellos es el presentado en [8].

El **soporte del lenguaje de programación** para movilidad posibilita una variedad de opciones, debido a que estos sistemas tienen un control casi completo sobre la implementación de la aplicación en tiempo de ejecución facilitando así el uso de checkpoints (y por lo tanto migración). No obstante mantener los canales de comunicación no es una tarea sencilla [9].

Los **agentes móviles** están siendo cada vez más populares debido a su estrecha relación con la Web. Su uso en la Web enfatiza la seguridad más que la complejidad, performance, transparencia y heterogeneidad. Los agentes móviles se implementan sobre lenguajes seguros, como por ejemplo Java, Telescript y Tcl/Tk.

3 Características

Esta sección trata los temas concernientes a la migración de procesos tales como complejidad, performance, transparencia, tolerancia a fallas y heterogeneidad. Estas características influyen directamente en la efectividad y el desarrolo de la migración de procesos.

3.1 Complejidad y Soporte del Sistema Operativo

La complejidad de la implementación y la dependencia con el sistema operativo son dos de los obstáculos que se oponen a la popularidad de la migración de procesos. La migración puede clasificarse según el nivel donde se la implemente. Puede implementarse como parte del kernel del sistema operativo, en espacio de usuario, como parte de un entorno de sistema, o como parte de la aplicación. Implementaciones a distintos niveles determinan diferente performance, complejidad, transparencia y reusabilidad.

La migración a nivel de usuario generalmente produce implementaciones simples pero también pobres en performance y transparencia como para ser de uso general para distribución de carga. Cuando la migración es implementada como parte de una aplicación se pierde reusabilidad, pues luego podría ser necesario modificarla. Otro inconveniente es que se necesita duplicar los mecanismos de migración en cada subsiguiente aplicación. Una mejora a esta estrategia es organizar el soporte para la migración en una librería reusable. Cuanto más bajo sea el nivel donde se la implemente mayor será la performance, transparencia y reusabilidad.

Sin embargo, la migración a nivel de usuario también tiene beneficios: las capas más cercanas a la aplicación tienen mayor conocimiento acerca del comportamiento de la misma y por ende es más simple crear políticas de migración que permitan aumentar la performance. Esta motivación es la que promovió el desarrollo de microkernels tales como los de Mach, Chorus y Amoeba, los cuales movieron mucha de su funcionalidad desde el kernel hacia espacio de usuario.

Los kernels extensibles, como los de Spin, Exokernel y Synthetix tomaron una alternativa diferente, permiten importar en el kernel partes implementadas por el usuario. Tanto los microkernels como los kernels extensibles brindan la posibilidad de extraer el estado de un proceso.

En los sistemas operativos tipo UNIX se deben modificar varios subsistemas del kernel para obtener soporte para los archivos abiertos y el manejo de señales. Los sistemas con pasaje de mensajes requieren un esfuerzo significativo para soportar el manejo de mensajes [10]. Los sistemas operativos actuales proveen mucho de este soporte, por ejemplo en cuanto a IPC distribuido con forwarding de mensajes. No obstante la migración todavía reta (y frecuentemente vence) a estos mecanismos [7].

3.2 Performance

El segundo factor que afecta la popularidad de la migración de procesos es la performance. El costo inicial (transferencia de estado) y el costo en tiempo de ejecución (transferencia de estado por demanda) provocado por la migración impacta en la performance. Ambos tipos de costo pueden ser significativos, dependiendo de las características de la aplicación y de la relación entre estado transferido de antemano (*eagerly*) vs. por demanda (*lazily*).

Si solamente se transfiere una parte del estado hacia otro nodo, la tarea puede ejecutarse antes y por lo tanto el costo inicial de migración es menor. Este principio se conoce como

evaluación perezosa: las acciones se efectuán cuando son realmente necesarias, más aún, se espera jamás tener que realizarlas.

3.3 Transparencia

Para lograr transparencia se debe evitar que tanto la tarea migrada como las demás tareas del sistema noten la migración, con la posible excepción de la caida en la performance. La comunicación con un proceso migrado puede retrasarse durante la migración pero no puede perderse ningún mensaje. Después de la migración el proceso debe ser capaz de continuar comunicandose a través de los mismos canales de I/O.

La transparencia puede ser soportada de varias formas, dependiendo del sistema operativo subyacente. Por ejemplo, Sprite y MOSIX mantienen una noción de un nodo home, en cambio Charlotte soporta IPC a través de links.

Un sistema con transparencia también asume que la instancia migrada puede ejecutar todas las llamadas a sistema como si no hubiese migrado. Este supuesto no siempre puede asegurarse debido a que existen sistemas implementados a nivel de usuario dondo no están permitidas las llamadas a sistema que generan señales hacia nodos remotos.

3.4 Tolerancia a Fallas

La tolerancia a fallas muchas veces fue mencionada como uno de los beneficios de la migración de procesos, sin embargo esta afirmación nunca fue respaldada con una implementación práctica.

Las fallas juegan un rol importante en la implementación de migración de procesos pues pueden ocurrir en un nodo fuente, un nodo destino o en el medio de comunicación. Existen esquemas de migración que son más o menos sensibles a cada uno estos tipos de fallas. Las dependencias residuales tienen un impacto particularmente negativo sobre la tolerancia a fallas, por lo tanto usarlas es un *trade-off* entre eficiencia y confiabilidad.

La tolerancia a fallas puede mejorarse de varias formas. Por ejemplo se puede reducir el impacto de las fallas durante la migración si se mantiene el estado del proceso en los nodos fuente y destino hasta que la instancia del nodo destino se convierte en un proceso normal y se informa de esto al nodo fuente. Una falla en un nodo fuente puede superarse si se "desconecta" completamente la instancia del nodo fuente una vez que fue migrada, sin embargo esto impide la aplicación de técnicas de evaluación perezosa. Una forma de evitar las dependencias residuales de comunicación es empleando técnicas de ubicación, como multicasting (en kernel V), o tolerancia en el nodo home (en Sprite y MOSIX). De esta forma las dependencias se agrupan en nodos dedicados, en lugar de estar dispersos en los nodos visitados, como es el caso de Charlotte, en el que periódicamente se colapsan punteros de redireccionamiento (y por lo tanto se reducen las dependencias funcionales) para el caso de la *garbage collection*.

3.5 Escalabilidad

La escalabilidad de un mecanismo de migración de procesos está fuertemente ligada a la escalabilidad del entorno subyacente. Puede medirse con respecto a la cantidad de nodos del sistema, o a la cantidad de veces que puede migrar un proceso a lo largo de su tiempo de vida, o a la complejidad y tipo de los procesos (como el número de de canales/archivos abiertos y tamaño en memoria).

El número de nodos del sistema afecta la organización y el manejo de las estructuras necesarias para mantener el estado residual del proceso y la denominación de los procesos migrados.

Algunos sistemas no son escalables en cuanto al número de migraciones que un proceso puede atravesar, esto depende del algoritmo y de las técnicas de migración empleadas. Este es por ejemplo el caso de Mach, donde en ciertas ocasiones el estado del proceso crece en tamaño al aumentar el número de migraciones del proceso. Esta situación es aceptable para un número pequeño de migraciones, pero en otros casos el estado adicional puede dominar el costo de migración y puede llevar a un mecanismo de migración poco útil.

Los algoritmos de migración deben evitar crear dependencias lineales en la cantidad de estado a transferir. Por ejemplo, en la estrategia *eager* de transferencias de datos, los costos son proporcionales al tamaño del espacio de direccionado, incurriendo en costos significativos para grandes espacios de direccionado. Los costos para un proceso copiado perezosamente son idependientes del tamaño del espacio de direccionado, pero pueden depender de la granularidad y del tipo del espacio de direccionado. Por ejemplo transferir un gran espacio poco denso puede tener costos proporcionales a la cantidad de regiones contiguas de espacios de direccionado, debido a que cada región tiene metadatos asociados a ella que deben transferirse en el momento de la migración.

Los canales de comunicación también pueden afectar la escalabilidad. Redirigir la comunicación a un proceso es aceptable para un número pequeño de migraciones secuenciales, pero después de un número grande de ellas los costos de redireccionamiento pueden ser significativos. En este caso puede emplearse alguna otra técnica, como por ejemplo la de actualizar los enlaces de comunicación.

3.6 Heterogeneidad

En las primeras implementaciones de migración el tema de la heterogeneidad no era considerado, por el contrario la homogeneidad era un requerimiento. En este momento esto no representaba una limitación considerable pues la mayoría del trabajo era realizado en clusters de nodos homogéneos.

Sin embargo, el reciente crecimiento de la computación mundialmente distribuida aumentó el interés de la migración heterogénea. Para lograr heterogeneidad el estado debe almacenarse en una representación independiente a la arquitectura. En general una aplicación se compila por adelantado para cada arquitectura, y se agrega código para poder conocer que procedimientos y variables existen en cada momento, y además se identifican puntos en los cuales la aplicación puede interrumpirse (checkpoints).

En los sistemas más recientes, la heterogeneidad es provista a nivel de lenguaje, como por ejemplo empleando la representación intermedia (*byte code* de Java), o utilizando lenguajes de scripts, como Telescript y Tcl/Tk.

4 Justificación de la Falta de Popularidad de la Migración

En esta sección trataremos de identificar las barreras que evitan que se adopte popularmente a la migración de procesos y que hacer para superarlas.

4.1 Prejuicios

Muchas veces la migración de procesos fue tratada como un ejercicio académico con pocas probabilidades de ser adoptada popularmente [6]. Para justificar esta postura se ha dicho que

la migración presenta:

- complejidad significativa,
- costos inaceptables,
- falta de soporte para la transparencia,
- falta de soporte para la heterogeneidad.

De hecho, aunque algunas implementaciones reforzaron estas creencias se seguió trabajando en migración de procesos. Más aún, recientemente se han visto cada vez más intentos de proveer migración y otras formas de mobilidad [11]. Se han implementado también sistemas de checkpoint/restart para soportar procesos con prolongados tiempos de ejecución. Finalmente, se está investigando el tema de agentes móviles en la Web.

Si analizamos las implementaciones existentes veremos que existen soluciones técnicas para cada uno de estos problemas (complejidad, costo y falta de transparencia y de heterogeneidad). La migración fue soportada con distintos niveles de complejidad: como parte de los mecanismos del kernel, como mecanismos a nivel de usuario, e incluso como parte de una aplicación. El tiempo ncesario para migrar un proceso fue reducido de segundos o minutos hasta milisegundos. También se introdujeron técnicas para optimizar la transferencia de estado[12]. Finalmente, tanto la transparencia como la heterogeneidad fueron alcanzadas (hasta cierto punto) en sistemas recientes.

4.2 Barreras a la Migración

Existen impedimentos reales a la popularidad de la migración:

- **Falta de aplicaciones.** Las aplicaciones científicas y académicas, (e.g. `pmake` y simulaciones) representan un pequeño porcentaje del total de las aplicaciones actuales. La mayor proporción está compuesta por las aplicaciones standard de PC, tales como los procesadores de texto, planillas de cálculo, juegos y programas de diseño, las cuales no se benefician significativamente con la migración.
- **Falta de infraestructura.** No hay ningún sistema operativo distribuido que esté siendo utilizado globalmente. Pocos de los sistemas operativos con características distribuidas desarrollados en el ámbito académico han sido transferidos al ámbito comercial, por lo tanto se debe aumentar el esfuerzo necesario para implementar migración de procesos.
- **La migración no es una necesidad de los usuarios.** Los usuarios se conforman con soluciones alternativas, incluso aún aunque no presenten el comportamiento uniforme de la migración, pues ellas son más simples y siguen propuestas conocidas. Por ejemplo invocación remota y acceso a datos remotos.
- **Factores sociológicos.** En el modelo workstation, cada nodo pertenece a un usuario y es lógico pensar que éste no estará de acuerdo con que procesos ajenos se ejecuten en su máquina. Para resolver este problema puede emplearse algún mecanismo de evasión de procesos como en Sprite, o se puedee reducir la prioridad de los procesos remotos, como en Stealth.

4.3 Soluciones

Generalmente pasa mucho tiempo hasta que una buena idea pasa del ámbido académico al comercial. Ejemplos de esto son la programación orientada a objetos, el multithreading, Internet y puede también ser éste el caso de la migración de procesos.

Desarrollaremos aquí cada una de las barreras identificadas en la subsección anterior y trataremos de predecir como encajará la migración dentro de las necesidades futuras. Obviamente esta subsección es especulativa debido al intento de extrapolar las necesidades del mercado y la tecnología.

- **Aplicaciones.** Para que la migración obtenga un lugar en el mercado se necesita una aplicación capaz de proveer una razón que obligue a los vendedores de sistemas operativos comerciales a suministrar los recursos necesarios para poder implementar y soportar migración de procesos. Los tipos de aplicaciones que pueden aprovechar la migración incluyen a las tareas que hacen uso intensivo del procesador (e.g. compiladores paralelos y simulaciones) y tareas dominadas por I/O que se beneficien del movimiento de un proceso hacia los datos o hacia otro proceso. Este tipo de aplicaciones es extremadamente raro en el ambiente de las computadoras personales, donde predominan procesadores de texto, planillas de cálculo y juegos. Sin embargo, las aplicaciones se están volviendo cada vez más distribuidas, modulares y dependientes a datos externos; y en un futuro próximo, debido a la creciente performance de las redes, será mucho más relevante ejecutar (migrar) aplicaciones a nodos cercarnos a los datos. Además, la modularidad facilitará la paralización de las aplicaciones (e.g. Java Beans y Microsoft DCOM).

- **Infraestructura.** Algunas versiones de sistemas operativos basados en UNIX (e.g. Linux, Solaris y FreeBSD) y Microsoft Windows 2000 se están conviertiendo en un standard *de facto*. Estos sistemas están comenzando a tratar las necesidades de *clustering* y de multicomputadoras de gran escala, ambos entornos propicios para la migración. Mucha de la infraestructura faltante está siendo incluida en los sistemas operativos comerciales o en sus entornos de programación.

- **Conveniencia vs. necesidad (impacto de la tecnología).** Las siguientes tendencias tecnológicas pueden influir en la migración: redes de alta velocidad, sistemas de gran escala y la popularidad de los *gadgets* móviles (e.g. versiones futuras de teléfonos celulares). Debido al aumento en la velocidad de las redes, la diferencia entre ejecución remota y migración se vuelve mayor. La capacidad de mover procesos durante su ejecución (pues por ejemplo, se notó que existe mucha comunicación remota) puede mejorar significativamente la performance. Además, como en los sistemas de mayor escala las fallas son más frecuentes, es necesario poder continuar la ejecución de un proceso en un nodo remoto. Otro punto interesante que hace atractiva a la migración es su uso en aplicaciones críticas o de prolongado tiempo de ejecución. Finalmente podemos mencionar la creciente popularidad de los gadgets móviles los cuales necesitarán soporte en software, e.g. migrar aplicaciones desde una máquina de escritorio hacia una notebook y eventualmente hacia un gadget.

- **Factores sociológicos.** El significado y la importancia del concepto de la workstation perteneciente a alguien está disminuyendo. Existen tantas computadoras hoy en día que los ciclos de CPU son cada vez más baratos. De hecho, existen muchos servidores que no pertenecen a un único usuario. Por otra parte es claro que el mundo se está volviendo

cada vez más "conectado", no es extraña la idea de correr código de alguien más en la workstation de uno. Este tema vuelve interesante y necesaria la investigación acerca de las cuestiones de seguridad.

Para resumir podemos decir que no creemos que sea necesario algún desarrollo revolucionario en el campo de la migración de procesos para que sea adoptada popularmente. Es cuestión de tiempo, desarrollo tecnológico y nuevas necesidades por parte de los usuarios para que se inicie un uso extensivo de la migración de procesos.

5 Resumen e Investigaciones Futuras

En este paper hemos caracterizado distintos mecanismos de migración de procesos. Además hemos tratado de identificar tanto algunas ideas preconcebidas así como también impedimentos reales a la popularidad de la migración.

Creemos que existe un futuro para la migración de procesos. Diferentes corrientes de desarrollo pueden llevar a popularizar la migración. A continuación enumeraremos algunas de estas posibles tendencias.

Una posibilidad es la de ubicar los mecanismos de migración a nivel del usuario, tratando de lograr una performance comparable a la de sistemas cuyos mecanismos están implementados a un nivel más bajo pero sin tener tantas complicaciones. El modelo checkpoint/restart está relativamente difundido y paquetes como Condor, LSF y LoadLeveler están siendo utilizados en aplicaciones batch y científicas en ambientes de producción, donde la demanda por los recursos de computación es alta y es posible aprovechar el balance de carga.

Un segundo camino involucra a los clusters de workstations. Los avances recientes en las redes de alta velocidad (e.g. ATM, Myrinet) redujeron el costo de la migración, permitiendo aún el desarrollo de implementaciones costosas.

Otra alternativa, esta vez más cercana a los usuarios de la vasta mayoría de las computadoras de hoy en día (MS Windows sobre plataformas Intel) llevaría la migración a los hogares y oficinas. Sun recientemente puso a disposición de las computadoras personales su arquitectura Jini y probablemente muchas otras compañias harán lo mismo. Actualmente uno puede imaginar un proceso migrando desde uno computadora hacia otro dispositivo en el mismo dominio, similar a la migración de agentes empleada en la Web.

Un cuarto argumento a favor de la migración de procesos, una vez más para aprovechar la vasta capacidad de procesamiento de un amplio rango de máquinas, consiste en realizar algún tipo de esfuerzo nacional o internacional de computación. Por ejemplo, en 1991 se sugirió que el gobierno de China podría resolver problemas complejos (e.g. factoreo de enteros largos) utilizando el poder de procesamiento de los televisores. Si fuera necesario, la migración de procesos podría proveer el mecanismo subyacente para la migración a gran escala sobre un conjunto cambiante de computadoras.

Finalmente, lo más promisorio es el uso de los agentes móviles en la Web. En este sentido, en lugar de los modelos de workstations y pools de procesadores el entorno de Web conecta computadoras como interfaces al modelo "la red es la computadora" perdiendo entonces relevancia la transparencia. Con respecto a la performance, ésta pasa a depender de la latencia de la red y por lo tanto la transferencia de estado no es tan importante como en el caso de las redes locales. Así es como siempre y cuando el entorno de ejecución sea seguro, los usuarios se verán estimulados a permitir la ejecución de procesos remotos en sus computadoras. Otro tópico importante a tener en cuenta es la heterogeneidad, la cual es automáticamente soportada

a nivel del lenguaje. De esta manera los agentes móviles en la Web solucionan impedimentos análogos a los de la migración de procesos posibilitando el uso masivo de ambas tecnologías, dado que comparten similares técnicas.

A lo largo de este trabajo vimos que se han realizado grandes avances en la migración de procesos, sin embargo los resultados obtenidos en cuanto a aplicabilidad y performance distan mucho de ser óptimos. Este estímulo sumado a lo interesante y promisorio que resulta el tema, vuelve a la migración de procesos un área de investigación no solamente ideal sino necesaria. Es nuestra intención seguir explorando en este campo y ser capaces de crear un sistema con soporte para la migración a partir de un nuevo sistema de memoria compartida distribuida implementada en software.

Referencias

[1] T. E. Anderson. A Case for Networks of Workstations. *IEEE Micro 15*, 1, 54–64, 1995.

[2] T. L. Casavant, J. Kuhl. Efects of Response and Stability on Scheduling in Distributed Computing Systems. *IEEE Transactions on Software Engineering*, SE-14(11), 1578–1588, 1988.

[3] M. Harchol-Balter, A. Downey. Exploiting Process Lifetime Distributions for Dynamica Load Balancing. *Proceedings of ACM Sigmetrics 1996 Conference on Measurement and Modeling of Computer Systems*, 13–24, 1997.

[4] A. Svensson. History, an Inteligent Load Sharing Filter. *Proceedings of the 10th International Conference on Distributed Computing Systems*, 546–553, 1990.

[5] C. Wang. Intelligent Job Selection for Distributed Scheduling. *Proceedings of the 13th International Conference on Distributed Computing Systems*, 288–295, 1993.

[6] N. Shivaratri, P. Krueger. Load Distributing for Loally Distributed Systems. *IEEE Computer*, 33–44, 1992.

[7] F. Douglis, J. Ousterhout. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software-Practice and Experience 21*, 8, 757–885, 1993.

[8] R. Zajcew, P. Roy. An OSF/1 UNIX for Massively Parallel Multicomputers. *Proceedings of the Winter USENIX Conference*, 449–468, 1993.

[9] E. Jul. Migration of Light-weight Processes in Emerald. *IEEE Technical Committee on Operating Systems Newsletter 3*, 1, 20–23, 1989.

[10] Y. Artsy, R. Finkel. Designing a Process Migration Facility: the Charlotte Experience. *IEEE Computer*, 47–56, 1989.

[11] P. Smith, N. Hutchinson. Heterogeneous Process Migration: the Tui System. *Software-Practice and Experience 28*, 6, 611–639, 1998.

[12] E. Roush, R. Campbell. Fast Dynamic Process Migration. *Proceedings of the 16th International Conference on Distributed Computing Systems*, 637–645, 1996.